UNIVERSITY OF QUEENSLAND
COMPUTER CENTRE

COMPUTER CENTRE

BULLETIN

Volume 2, Number 3.
3rd March, 1969.

Editor:
H. L. Smythe.

EDITORIAL COMMENT

It is proposed to introduce in future Bulletins a *"Letters to the Editor"* feature. This could include letters from the various computer users regarding interesting or unusual programming difficulties they might have encountered. If desired, the letter would be given to one of the Programmers for perusal. Both the letter and his observations would then be published in the Bulletin.

Because the success of a scheme of this kind depends very much on the contributions received from the readers, the Editor will sincerely welcome all such letters. In this way, it is hoped that the *"Letters to the Editor"* feature will become established as a regular part of the Bulletin.

Should any reader have an article which he considers to be too long for inclusion in the *"Letters to the Editor"* feature, it could be published in the Bulletin as a separate contribution under the author's name.

This month's Bulletin contains another in the series of articles on the staff of the Computer Centre, introducing two Lecturers in Computing and the Technical Writer. The section on Library Accessions is continued, and Mr. Oliver has produced an interesting and informative article on multiprocessing.

STAFF OF THE COMPUTER CENTRE

INTRODUCING TWO LECTURERS IN COMPUTING

Christopher ('Chris') de Voil is a Lecturer in Computing (Systems Programming) within the Department of Computer Science.

In 1960, Chris graduated from the University of Queensland and spent a further year in the Department of Electrical Engineering where he was engaged in post-graduate research. In 1961, he joined the Aeronautical Research Laboratories, Department of Supply, where he worked as an Engineer and then Research Scientist, until 1967. Initially during this period, he was involved in the analysis and design of control system elements and allied electronic equipment. Later, however, he joined the computer group where he worked in the design, development and application of analogue, hybrid and digital computing systems.

Graduate Administrative Assistant, who commenced duty on 2nd January, 1969. Helen, who holds a Bachelor of Arts degree from the University of Queensland, is now the Technical Writer for the Computer Centre.

Her duties include the preparation and editing of the Bulletin and other Computer Centre publications, and generally assisting the remainder of the staff with the production of technical writeups, memoranda, and research reports.

FORTRAN IV SYSTEM MODIFICATION

TANH SUBROUTINE

This routine returned an incorrect value and has now been changed to work correctly.

Previously, index registers were stored by TANH in absolute suffix locations FSAVE 1 and 2. These locations were subsequently used by EXP (called by TANH), resulting in the failure of TANH to return properly to the calling program.

LIBRARY ACCESSIONS

This section of the Bulletin is an attempt to keep readers informed about the books and periodicals, in the computer field, which have recently been acquired by the University of Queensland Library. As they are selected from the University of Queensland Library Accession List by title only, there is no guarantee that the material is of high quality. It is hoped, however, that the list may prove to be a useful guide to readers.

The following list comprises the accessions for the month of September, 1968.

Bartee, Thomas C. *Theory and Design of Digital Machines.* [c 1962] (510.7834 BAR, Engin. Lib.)

D'Hoop, H. *SAHYB (Simulation of Analogue and Hybrid Computers).* 1965 (Q 519.92 DHO, Engin. Lib.)

Krishnamoorthi, B. *Time-Shared Computer Operations With Both Interarrival and Service Times Exponential.* 1965 (Q 519.92 KRI, Engin. Lib.)

Smith, Fred W. *Estimation of the Laplace-Transfer Function from Sampled Data.* 1965 (Q 517.352 SMI, Engin. Lib.)

The Australian Computer Journal. 1967/68 and onwards (519.92 AUS Periodical Stack, Main Library)

Simulation. V. 10, 1968, and onwards. (519.8 SIM, Engin. Lib.)

Mize, Joe H. *Essentials of Simulation.* 1968. (620.7 MIZ, Engin. Lib.)

Orr, William D. comp. *Conversational Computers.* 1968. (621.381958 ORR, Engin. Lib.)

Stewart, Charles A. *Basic Analogue Computer Techniques.* [c 1967] (621.381957 STE, Engin. Lib.)

Berkeley, Edmund C. ed. *The Programming Language LISP.* 1964. (651.8 BER, Engin. Lib.)

Bycer, Bernard B. *Digital Magnetic Tape Recording: Principles and Computer Applications.* [c 1965] (651.263 BYC, Engin. Lib.)

Gregory, Robert H. *Business Data Processing and Programming.* 1960. (651.8 GRE, Main Lib.)

International Business Machines Corporation. *IBM Data Processing Techniques.* 196-. (Q 658.502 INT, Accountancy Seminar Room)

Rosen, Saul. ed. *Programming Systems and Languages.* [1967] (651.8 ROS, Engin. Lib.)

Rosove, Perry E. *Developing Computer-Based Information Systems.* [1967] (658.502 ROS, Main Lib.)

Schwartz, Jules I. *Observations on Time-Shared Systems.* 1965. (Q 651.84 SCH, Engin. Lib.)

Sollenberger, Harold M. *Major Changes Caused by the Implementation of a Management Information System.* 1967. (658.502 SOL, Main Lib.)

Yeomans, Athol. *It Figures! An Introduction to Computers for Management.* 1966. (651.8 YEO, Accountancy Seminar Room)

The following list specifies the accessions for the month of October, 1968:

Singh, Jagjit. *Great Ideas in Information Theory, Language and Cybernetics.* 1966. (001.53 SIN, Engin. Lib.)

Hays, David G. *Computational Linguistics*. 1967. (Q 016.41 HAY, Engin. Lib.)

Advanced Seminar on the Spectral Analysis of Time Series. University of Wisconsin, Madison, Wisc. Oct. 3-5, 1966. 1967. (519.1 ADV, Maths. Lib.)

Barbieri, R. *Computer Compiler Organization Studies*. 1967. (Q 519.92 BAR, Engin. Lib.)

Bucy, R.S. *Optimal Filtering for Correlated Noise*. 1966. (Q 519.92 BUC, Engin. Lib.)

Coffman, E.G. *Interarrival Statistics for TSS*. 1965. (Q 519.92 COF, Engin. Lib.)

Computing Technology Ing. *Survey of Computer-Program Documentation Practices at Seven Federal Government Agencies*. 1967. (Q 519.92 COM, Engin. Lib.)

Dineley, Jack L. *A Manual of KALDAS Programming*. 1967. (519.92 DIN, Engin. Lib.)

Funk, James E. *"Slash" ALGOL Simulated Hybrid Computer*. 1965. (Q 519.92 FUN, Engin. Lib.)

Hemmerle, William J. *Statistical Computations on a Digital Computer*. [c 1967] (519 HEM, Soc. and Prev. Med. Dept.)

Linde, Richard R. *Operational Management of Time-Sharing*. 1966. (Q 519.92 LIN, Engin. Lib.)

McKeeman, William M. *An Approach to Computer Language Design*. 1966. (Q 519.92 MACK, Computer Centre)

Annual Review of Automatic Programming. 1960. (510.78 ANN, Engin. Lib.)

Computer Studies in the Humanities and Verbal Behaviour. 1968, and onwards. (510.7834 COM, Periodical Stack, Main Lib.)

Bryan, G.E. *JOSS: User Scheduling and Resource Allocation*. 1967. (Q 621.38195 BRY, Engin. Lib.)

Doncav, Boris. *Computer Technology*. 1966. (Q 621.38195 DON, Engin. Lib.)

Enslow, Philip H. *Documentation Techniques for Digital Hardware*. 1967. (Q 621.381958 ENS, Engin. Lib.)

Laver, F.J.M. *Some Developments in Computing*. 1967. (Q 621.38195 LAV, Main Lib.)

Stagg, Glenn W. *Computer Methods in Power System Analysis*. 1968. (621.31018 STA, Elect. Engin.)

Borko, Harold. *Automated Language Processing*. 1967. (651.8 BOR, Main Lib.)

Joint Technical Committee on Terminology. *IFIP-ICC Vocabulary of Information Processing*. 1966. (651.8 JOI, Engin. Lib.)

U.S. Defense Intelligence Agency. *Manual of Data Processing Standards.* 1966. (Q 651.8 UNI, Engin. Lib.)

Commonwealth Scientific and Industrial Research Organization. Division of Computing Research. *Technical Note No. 24.* 1968, and onwards. (651.8 COM, Engin. Lib.)

Kjaer, Jorgen. *Calculation of Ammonia Converters on an Electronic Digital Computer.* 1963. (660.283 KJA, Engin. Lib.)

Symposium on Cost Control of Projects, London, 1966. *Cost Control of Projects.* 1966. (Q 660.28 SYM, Engin. Lib.)

MULTIPROCESSING CONCEPTS

I. Oliver

Multiprocessing may be defined as the operation of a computer system which has more than one central processing unit (processor). Until recently, computer systems had only one processor which meant that only one stream of calculation could be processed at a time. Now it is possible to link processors together so that multiple streams of computation can occur simultaneously. This results in an overall increase in service to the user by reducing turnaround time. As a simple example, one processor may process a long compute-bound job while another may service a sequence of debug runs.

Multiprocessing creates some new difficulties for systems programmers. The scheduling of jobs and the allocation of other resources such as card readers and printers require a considerable amount of machine language code. The PDP 10 computer at the University of Queensland has two processors and, when multiprocessing is fully operational, more storage capacity than the entire GE 225 computer memory will be reserved for the operating system programs.

To understand what multiprocessing involves, let us consider the "ancient" computers of the 1950's, taking as an example a very simple program which merely reads some numbers from cards and prints them on a printer, thus:

```
          DO 10 I = 1, ∞
          READ, A, B, C
   10     PRINT, A, B, C
```

When executing this program, the computer alternately operates the reader and the printer to read a card or print a line.

Although this arrangement is attractively simple, there are two major problems of efficiency in these old computers:

(i) the processor must wait for the peripheral device (such as card reader or printer) to complete its task before it can proceed to the next instruction in the program;

(ii) the peripherals cannot run at full speed since only one peripheral can be operating at any one time.

We then ask the question - can the computer be built to read the *next* card *while* the processor works on the last card, and to print the *last* line while the processor works on the *next?*

In the old system the computer is organized like this:

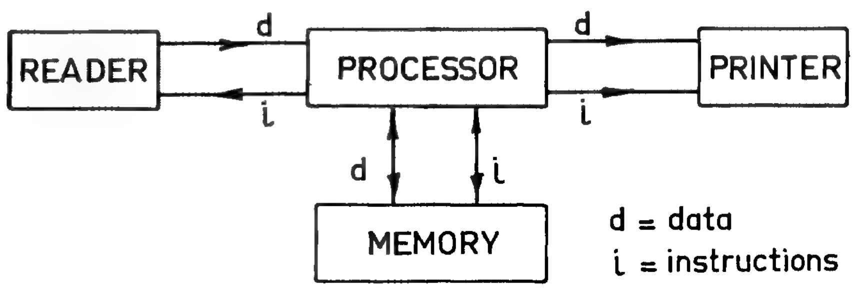

Figure 1

Note that the processor is central to the whole operation. It can do only one thing at once, i.e. read, print, or compute. It must store each column of the card in memory and must send each character to the printer since the peripherals have no direct access to memory.

A major advance was the introduction of device controllers which have direct access to memory. A computer with controllers may have a structure such as:

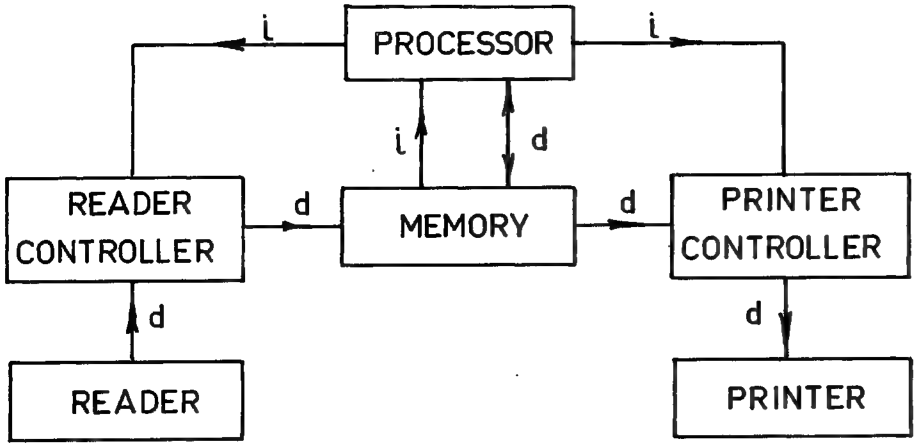

Figure 2

Now the system is more complicated, and more difficult to program. But the processor can execute instructions *while* cards are being read directly into memory by the reader controller, and *while* lines are being printed by the printer controller. Note that the processor merely *initiates* reading and printing. The controllers take over and carry out the actual data transfer.

By proper programming, the card reader may be made to read cards continuously into a number of blocks of memory called *buffers*. The processor waits only for the first card to be read. It initiates the reading of the next card and then carries out the desired computations on the first card. Similarly, the printer can lag behind the processor, autonomously printing from another string of buffers being filled by the processor.

Two problems arise. Suppose the processor has very little computing to do on each card. It will therefore try to get ahead of the reader. The program must give an instruction which tests the status of the reader and makes the processor wait until the next card has been read.

Alternatively, the processor may have a considerable amount of computing to do so that the reader may tend to get ahead of the processor. In this case, the reader must be stopped by the processor when all of the available buffers are full. At some later time when the processor has used the information in some of the buffers, the reader can be restarted. A similar situation occurs with the printer.

Operation of the peripherals in this manner is called *concurrent* or *asynchronous*. Nearly all modern computers have concurrent operation of peripherals which brings a great increase in efficiency.

If a batch stream running on such a computer is observed, it is clear that further economies may be achieved. Some of the jobs require a large amount of processor time and use the peripherals scarcely at all, while others may process large masses of data with relatively little computation involved. While the first type of program is being run, the processor is being used at optimum efficiency (disregarding the ability of the programmer to write programs), but the peripherals are being used most inefficiently. The reverse is true in the second case.

The technique of *multiprogramming* may help to improve the efficiency. Multiprogramming consists of the operation of a computer system in which one or more independent programs are run simultaneously. Ideally, programs which are compute-bound should be run with one or more programs which require little processor time and make extensive use of the peripherals. There are two main requirements:

(i) both programs must fit into core memory together;

(ii) the programs must not use the same input/output devices at the same time.

Even if these requirements are met, there is another problem. The processor can service only one of the programs at a time. Although the compute-bound job requires most of the processor time, the other jobs must use it occasionally to initiate input/output and to carry out a small amount of processing.

Multiprogramming computers enable the processor to be switched from one program to another in core so that all programs are serviced. One technique is to let the input/output-bound jobs have the processor long enough to drive the peripherals as fast as possible. The remaining time can be used by the compute-bound jobs.

We see then that the compute-bound jobs must be interrupted whenever any input/output must be carried. Special hardware must be available to enable these interrupts to occur automatically.

Even with multiprogramming, the programs in core are not run in a completely simultaneous and asynchronous mode. The processor is capable of

processing only one job at a time. The main point is that the processor advances a number of programs in successive time slices so that, at the macro level, it is not necessary for one job to be complete before another can be commenced.

If multiprogramming is to be a viable proposition for a computer service bureau, several other requirements must be met:

(i) *Independence of preparation of programs.* A programmer should not be required to know which programs will be run concurrently with his program and to take special action to enable it to be so run.

(ii) *Minimum information from the programmer.* A programmer should not be *required* to supply any additional information to allow his program to be multiprogrammed with other jobs. However, an exception may be made if such additional information allows the system to process his job more efficiently.

(iii) *Non-interference.* No program should be permitted to introduce errors into other programs. For example, a program should not be able to access any of the files or core memory of any other job. Since instructions in a program may modify themselves during execution, dynamic memory protection is required. Special hardware is necessary to check the address of every instruction prior to the execution of that instruction. In addition, if a program is stuck in a loop, it should be removed from the system automatically when its time limit expires.

How is it possible to multiprogram two jobs which require the same peripheral devices for input/output? The *input/output well* is the technique normally used for overcoming this problem. A program known as a *symbiont* is used to read all of the input for a given job onto a backup storage device such as a magnetic disk or drum. The whole file is transferred in this way before the program is executed. An input statement in the program then causes the disk or drum to be accessed rather than the original input device. Similarly, symbionts allow output to be stacked up on disk or drum. The sole purpose of the symbiont technique is to drive the peripherals as fast as possible.

The PDP 10 is capable of multiprogramming more jobs than can be fitted into core at one time. Jobs can be swapped out of core for a time onto the fixed-head disks and replaced by other jobs, all of them only partially executed. In general, swapping is a costly overhead, i.e. it wastes system resources.

But it does not allow more users to access the system than would otherwise be the case, and, in a less than fully loaded system, may not significantly affect service to individual users.

It would appear that, in a multiprocessor system with, say, two processors, the throughput could be increased by a factor of two. However, the actual factor is somewhat less than two due to certain overheads. A more complicated program is required in the operating system to schedule the resources of a multiprocessor system so that more processor time is used in executing code that is unproductive (from a user's point of view.) In addition, both processors must access core memory. A finite time called the cycle time (approximately one microsecond on the PDP 10) is required to read or store a word. Processors requiring access to memory at frequent intervals may be held up for a fraction of a microsecond by other processors also accessing memory.

Time-sharing is the simultaneous *interactive* access to a computer by a number of independent users. Given a multiprogramming system, it is possible for users to have immediate access to the computer facilities through remote terminals such as teletypewriters or cathode ray displays. Each user can initiate programs and can have almost immediate response on his terminal to its execution. The main point is that the access is interactive. Each user apparently has access to nearly all of the facilities of the machine as if he were suituated at its console. If there are n users, each user sees a machine roughly $\frac{1}{n}$th as fast as the actual machine. The figure is somewhat less than $\frac{1}{n}$ due to the processor time used in scheduling operations. But large time-sharing computers are so fast that the effective user speed is usually more than adequate.